

DATA COLLECTION AND EVALUATION FOR
EXPERIMENTAL COMPUTER SCIENCE RESEARCH

Marvin V. Zelkowitz
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

The Software Engineering Laboratory has been monitoring software development at NASA Goddard Space Flight Center since 1976. This report describes the data collection activities of the Laboratory and some of the difficulties of obtaining reliable data. In addition, the application of this data collection process to a current prototyping experiment is reviewed.

I. INTRODUCTION

There is a significant need to collect reliable data on software development projects in order to provide an empirical basis for making conclusions about software development methodologies, models and tools. However, such data is usually hard to collect and even harder to evaluate. Software is a multibillion dollar industry where 100% cost overruns are common, and maintenance activities can take up to 70% of the total cost of the system [11]. The availability of reliable data to evaluate competing software development techniques is crucial.

As Lord kelvin stated, "I often say that when you can measure what you are speaking about, and express it in numbers, you can know something about it, but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind." The lack of adequate measures is certainly a problem in the software industry today.

Many of the recent analyses of the software development process are based on data that is obtained from university experiments. Students often program special problems whose results are subjected to analysis. This gives the researcher the 10 to 100 data points necessary for statistical validity of the results. However, by virtue of being part of an academic program, such experiments are necessarily small and usually involve inexperienced programmers. There is a need to

extend the scope of these experiments to a level appropriate to the multibillion dollar industry.

Most software development data in industry has been collected after the fact. That is, a project is built and then a pile of documents are handed to a research group for evaluation. Often, critical information is missing and the results are not what one would expect. Rather than following the model of archeology - the study of dead software projects, software evaluation must model sociology - the study of living software societies. Data must be collected from ongoing projects, but the software sociologists must not impact the objects of their study. Given the need to finish projects on time and within budgets - a goal too often missed - it is difficult to justify spending money on data collection and evaluation activities.

Specifically to address these problems, the Software Engineering Laboratory (SEL) was set up within NASA Goddard Space Flight Center in 1976. The goal was to study software development activities within NASA and report on experiences that will improve the process. This report describes the SEL and its experiences over the last six years.

II. THE SOFTWARE ENGINEERING LABORATORY

In 1976 the SEL was organized to study software development within the NASA environment. More specifically, its primary charter was to monitor the development of ground support software for unmanned spacecraft. Each such system was typically 30,000 to 50,000 source lines of Fortran and took from 8 to 10 programmers up to two years to build. While this environment is not representative of all software development environments, SEL experiences are generalizable in some respects:

a) Ground support software includes several program types such as data base functions, real time processing, scientific calculations and control language functions. The software is largely implemented in Fortran.

b) By looking at a relatively narrow environment, data collected from many projects can be compared. Thus we get some of the benefits of a carefully controlled experiment without the expense of duplicating large developments. We do not have the problem of looking at a variety of

projects, like compilers, COBOL programs, ground support software, MIS programs and then trying to say something consistent about all of these.

To date, 46 projects have been studied, containing over 1.8 million lines of code. Over 150 programmers participated in these projects, and the data base contains over 40 million bytes of data. The general SEL strategy is to carefully monitor a project and regularly collect data during its development. The data is then entered in the SEL data base for analysis. The purpose of this report is not to dwell on specific research results based on this data (See, for example, [8] for a collection of published papers about the SEL) but is concerned with the problems of collecting data, and what we have learned from this process.

III. DATA COLLECTION

III.1 MODEL GENERATION

In order to fully take advantage of the available data, it must be known what information is desired. The models and measures that are to be investigated must be defined. A random data collection activity will usually miss relevant data, and then it will be too late to try and recover that information.

In the SEL, two classes of measures were identified for study, and the data collection activities were oriented around those areas. The initial activities included:

a) Process Measures. Evaluating personnel and computer resources over time was a clear need. One activity was to try and validate models that others have identified (e.g., the Putnam Norden Rayleigh curve [1]) while another activity was to try and build new models to fit the empirical data (e.g., the Parr curve [7]). Once models were identified, their predictive nature was studied as a means of resource scheduling.

The generation and correction of errors is another activity that has important economic consequences. However, few models are available to build upon, so there was a need to develop new models of errors and investigate their effects upon performance.

b) Product Measures. The size, structure, and complexity of software are other important economic factors to consider. The evaluation of measures such as the software science measures of Halstead [5], the cyclomatic complexity of McCabe [6] and other measures developed within the SEL was another early goal.

Reliability is a critical activity in most environments. In our particular environment, the software that was previously developed was highly reliable (typically under 10 errors in an operational system), so that reliability, while important, was not a primary driving force in organizing the SEL.

III.2 FORMS GENERATION

The first process in evaluating empirical data is the data collection activity. Ideally, you would like the process to be automated and transparent to the programmer. However, this was not possible in this situation. We were interested in the human activities of software development. Thus we needed detailed information about how programmers spend their time. Because of this, a decision made early in the life of the SEL was that some data would be manually collected using a series of forms.

There is a significant tradeoff consideration at this point. If we tried to collect too much information, programmers would object to the interference of the data collection activity on their work. If too little information was asked, then there would be little point in collecting it.

We first developed an initial set of reporting forms. These have been revised several times since then. Each time certain fields were clarified and the amount of information sought decreased somewhat. At the present time, the effort required to fill out the forms is not significant. Initially seven forms were developed. However, only three are used heavily. These seven forms are:

a) Resource Summary. This form lists the number of hours per week spent by all personnel on the project. This information is obtained mostly from the weekly time cards supplied by the contractor. It is easy to obtain this data, and causes little overhead to a project. However, it is very useful for monitoring global resource expenditures, especially in conjunction with the follow-

ing Component Status Report.

b) Component Status Report. This form is submitted weekly by each programmer. It lists for each component of the system (e.g., Fortran subroutine) the number of hours spent on each of nine categories (e.g., design, code, test, review, etc.). The detail required by this form initially caused some concern; however, in looking over past forms the average programmer worked on only 5 to 10 components per week and only 2 or 3 activities per component. Thus the overhead was not excessive. While the data is only approximate to the nearest hour, we believe that it is more accurate than many other data collection procedures.

For example, many research papers give percentages for design, code, and test on a project. However, these are usually taken from resource summary data and calendar date milestones. If a design review occurs on a Friday, then all activities up until that date are design, with all activities the next week being code. In the SEL environment, there was approximately a 25 percent error in using calendar dates for percent effort [4]. On four projects, approximately 25 percent of the design occurred during the coding phase, while almost half of the testing occurred prior to the testing phase (Figure 1). The Component Status Report is critical for a proper view of development activities.

c) Change Report Form. This form is completed after each change to a component is completed and tested. Due to the number of changes that a component undergoes during early development, there was no attempt to capture this data before the component was "complete" (i.e., through unit test). Note that we are capturing "changes" and not simply "errors." All modifications, due to errors or other considerations such as enhancements, are tracked.

Besides identifying the type of change, this form also identifies the cause of the change - they are not always the same, although programmers have difficulty separating the two. The form also asks for information on the time to find and correct an error, and what tools and techniques were used in the process.

In some environments, the introduction of this form might cause programmers to object; however, this was not the case in our environment. A standard change monitoring procedure was

in place, so we simply changed the form that this branch of NASA GSFC was using before the SEL was created.

These three forms provide the most important data collected by the SEL. Four other forms have been created and used with limited success. These are:

d) Component Summary. This form identifies the characteristics of each component in a system. It gives the size, complexity and interfaces. The goal was to have this form filled out at least twice - once when the component was first identified during design, and again when it was completed. Our experience was that the initial form was filled out before much relevant information was known, and the data on the final form could be extracted automatically from the source code data base.

e) Computer Run Analysis. An entry on this form is filled out for each computer run giving characteristics of the run (execution time, purpose of run, components processed) as well as whether the run met its objectives. This is one form that could be automated. However, the usual range of operating system "Completion Codes" is inadequate for many purposes. For example, a debugging run that was expected to fail at a certain statement, but ran to a successful exit, would have a satisfactory completion code, yet it was a failure as a run since the desired error did not occur.

An interactive job submittal system could help. Before any run, the system could prompt for some of this information. After the run, the system could ask what happened. Since the current NASA environment consists primarily of interactive editing with batch processing, such an online process would have been difficult to implement.

f) Programmer Analyst Survey. This form attempts to characterize the experiences of the programmers on the project in order to get a general profile of the project team. However, we immediately ran into confidentiality problems concerning personnel records. We never got the detailed information that we desired, but have obtained general comments on each programmer - although the goal is NOT to rate programmers. If there is any hint of any of this data being used for any sort of personnel action, then compliance drops sharply and the value of the data becomes

open to question.

g) General Project Summary. This is a form that provides a high-level description of a project. Since the software is developed by NASA and contractor personnel, the form is somewhat superfluous and the information is entered directly into the data base.

An important consideration in forms development is consistency in collecting data. Along with each form a detailed instruction sheet was developed, as well as a glossary of relevant terms like "component," "line of code," and "life cycle phase." For example, we chose the name "component" rather than "subroutine" or "module" simply because those terms were well known (with alternative meanings) and we did not want to evoke any preconceived but wrong image in the minds of the participants. Even so, there was a great deal of confusion about the meanings of the various terms. During the early days of the SEL, many meetings were held to explain the process to programmers. since each programmer worked about one year on a project, after six years there is a large core of personnel experienced in filling out our reporting forms..

III.3 DATA PROCESSING

After being filled out, each form is entered into a data base on a PDP 11/70 computer. In addition to the forms previously described, analyzers were run over the source programs to extract additional information, including lines of code and other measures such as the Halstead software science measures.

Another step in forms processing is data validation. Someone must review the forms as they are submitted. This is expensive, but necessary. It is a quick way to catch and correct errors. In addition, the data entry program should check for data consistency and value ranges. For example, if the program is to read in input in the format MMDDYY, then a month input that is not a number in the range from 01 to 12 must be rejected. A field requiring an input of A, B, or C should reject any other value. Even though we manually check each form, a validation program was more effective for catching errors.

All forms, especially the change report form, need to be reviewed by SEL personnel. Two common errors in the Change report form are to turn in one change report form which actually represented several errors, and the submission of multiple forms for the same error. From earlier work over half of the change report forms were modified following a careful study of each form. This is an expensive process, but needs to be done in order to have accurate data about your environment.

Redundancy of data is another important consideration. Collecting the same or similar data on multiple forms allows for cross validation. There should be a reasonable correlation between the collected values. The resource summary and component status reports have been the easiest to validate. The Computer Run Analysis form is important for validating some of the change report data; however, limited availability of this form has handicapped some of this validation work. Because of that, it is important to manually check each change report form for selected projects.

IV RESEARCH ACTIVITIES

IV.1 PREVIOUS RESEARCH

Research in the SEL has centered on resource and error models and on predicting software productivity. ([8] is a collection of relevant papers published over the last few years.) Perhaps the most important conclusion - although obvious in hindsight - which is relevant to this current discussion is that there is no typical software development environment.

All models include parameters - factors which represent variables in that environment (Figure 2 represents a list of factors from the SEL as well as two other studies [10] [3].) When models based on other environments are applied to the NASA environment, they invariably fail. Does that mean that NASA is different? unique? much better or much worse than other environments? For example, SEL programmers show much higher productivity in lines of code per week than in other organizations. Does that mean that other organizations should pirate away NASA's staff?

Perhaps, but another explanation becomes apparent when NASA's environment is studied in detail. In the SEL, most of the projects are similar ground support software systems. Thus the top level design for these projects are similar. Programmers are experts at this particular problem - thus high productivity. Many factors affecting requirements and design do not apply here. On the other hand, a contractor that bids on a variety of projects - an operating system, a compiler, a data base management system, an attitude orbit determination program, etc. does not build an institutional knowledge about any one particular environment. Requirements and design factors now become significant in this environment and productivity drops.

All companies operate in a different manner. Company policy as to working conditions, computer usage (batch or interactive), leave policy and salaries, management, support tools, etc. all affect productivity. Thus each organization (probably even separate divisions within a single organization) has a different structure and a different set of parameters.

For this reason, one must first calibrate any model to be applied. First develop a quantitative relationship using many factors. Chose those factors relevant to your environment. Calibrate the equations based upon previous projects, and then use the calibrated model for prediction [2]. It is this important calibration step that is missing from most models.

For example, if a baseline equation is given by:

$$\text{Effort} = a * \text{size} + b$$

then one can fit a and b from historical data; and the units of size can be determined from those relevant to your environment - such as lines of code, lines of source (including comments), number of modules, number of output statements, etc. Thus instead of a single model, there is a class of models tailored to each environment.

IV.2 PROTOTYPES

Over the past few years various methodologies have been studied by the SEL. A current SEL activity is the development of software prototypes. Currently software is designed, built and delivered. Rarely is the product evaluated in advance. However, the use of engineering prototypes

in a preliminary evaluation is starting to be discussed by software engineering professionals [9].

While the term is appearing with increasing frequency, what does it really mean? Is it a quick and dirty throw-a-way implementation or a carefully designed subset of a final implementation? What are the cost and reliability parameters for a prototype compared to a full implementation.

Currently data on the subject is meagre and usually based on small projects [12]. The SEL is now investigating a larger implementation with some techniques as applied to previous SEL projects.

Briefly, the target implementation is an integrated support system for flight dynamics research. Currently, experimenters (NASA scientists), in trying a new spacecraft model (e.g., a new orbit calculation) must understand the structure of the existing system, access the Fortran source modules, modify them, rebuild the operating program, test it, and then run the experiment - a complex and costly process. The new system is expected to "understand" several flight dynamics systems and to provide a higher level command language that guides the experimenter through the process of building a new version of a system, even if the experimenter is not thoroughly familiar with the existing system. This system is basically a command language interpreter with a complex data dictionary describing the underlying flight dynamics subsystems.

This program is quite different from existing software produced by NASA, so the plan is to prototype it first. Two classes of data will be obtained from the prototype:

a) Characteristics of the process. The Computer Science world has little information available about prototyping, thus this data will add to the general knowledge about this process. What does the life cycle of a prototype look like? How much time is spent in design? code? test? Are errors crucial or can they be side-stepped in the prototype somewhat by "eliminating" the offending feature in the requirements?

Similarly, how does prototyping effect the later full implementation? Will design be easier? Will productivity be higher? Will the overall cost of the system plus prototype be less than the cost of just the full system? Will reliability be higher or the interface more "user friendly?"

b) Predictive nature of the prototype. Once a prototype is built, is it successful? How does one measure success? Will the full system be successful based upon an evaluation of the prototype? A set of measures will be built into the prototype to provide some of these answers.

A baseline study will be made of how experiments are conducted - the cost of machine and people resources will be measured. Some of these experiments will be repeated with the prototype to derive a cost. These will be used to predict the cost of using the full system. If acceptable, then that design will be used for the full implementation, if not, then the design will be modified to correct the problem in the full implementation.

In addition, data will be collected on how often features are used in the prototype, and also how often the prototype is being circumvented in order to provide features that currently do not exist but are needed by the users.

Once the final system is built, the predictive model can be validated in order to aid in developing a theory of software prototypes.

V. CONCLUSIONS

The Software Engineering Laboratory has been in existence for six years and has studied almost 40 projects. The empirical data that has been collected supports several conclusions:

(1) Data collection is hard and expensive. It must be dynamically collected during the development of a project and not after completion.

(2) Data must be validated. Error rates on manually filled out forms are high. A lack of standardized nomenclature for the field hurts consistency. Much effort must go in training personnel to understand the data collection methodology.

(3) Each software development environment is unique. Baseline equations must first be calibrated with past projects before a model can be used in the future.

(4) Little is known, but much is being said, about software prototypes. The SEL is currently studying this issue as part of its ongoing activities.

VL ACKNOWLEDGEMENTS

This paper was supported by NASA grant NSG-5123 to the University of Maryland. The SEL is under the direction of Frank McGarry of NASA GSFC. Dr. Victor Basili directs the University of Maryland activities of the SEL, and Jerry Page is the coordinator for Computer Sciences Corporation. The results described in this report were developed by the author, the above mentioned persons, as well as several graduate students at the University of Maryland and researchers at CSC. The author is indebted to David Card of CSC for his detailed comments on an earlier draft of this paper.

VII. BIBLIOGRAPHY

- [1] Basili V. R. and M. V. Zelkowitz, Analyzing medium scale software developments, Third International Conference on Software Engineering, Atlanta GA, May 1978.
- [2] Basili V. R., Models and metrics for software management and engineering, ASME Advances in Computer Technology 1, January, 1980.
- [3] Boehm B., Software Engineering Economics, Prentice Hall, 1981.
- [4] Chen E. and Zelkowitz M. V., Use of cluster analysis to evaluate software engineering methodologies, Fifth International Conference on Software Engineering, San Diego CA, March, 1981.
- [5] Halstead M., *Elements of Software Science*, American Elsevier, 1977.
- [6] McCabe T., A complexity measure, *IEEE Transactions on Software Engineering* 2, 1976.
- [7] Parr F., An alternative to the Rayleigh Curve model for software development, *IEEE Transactions on software engineering* 6, 1980.
- [8] Collected Software Engineering Papers: Volume 1, SEL-82-004, Code 582.1, NASA GSFC, July, 1982.
- [9] ACM SIGSOFT Software Engineering Symposium: Workshop on Rapid Prototyping, Columbia, MD, April, 1982.
- [10] Walston C. and C. Felix, A method of programming measurement and estimation, *IBM Systems Journal* 16, No. 1, 1977.
- [11] Zelkowitz M. V., A. C. Shaw and J. D. Gannon, *Principles of Software Engineering and Design*, Prentice Hall, 1979.
- [12] Zelkowitz M. V., A case study in rapid prototyping *Software Practice and Experience* 10, 1037-1042, 1980.

BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu and D. S. Wilson, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

¹SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, September 1982

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, A. L. Green, and C. E. Goorevich, February 1980

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

¹SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

SEL-80-104, Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1), W. Decker and W. Taylor, December 1982

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

¹SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

¹SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, and F. E. McGarry, September 1981

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wyckoff, July 1983

¹SEL-81-003, Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, and G. Page, September 1981

SEL-81-103, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo and D. Card, July 1983

¹SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

¹SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

¹SEL-81-105, Recommended Approach to Software Development, S. Eslinger, F. E. McGarry, and G. Page, May 1982

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

¹SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

¹SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

SEL-81-110, Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics, G. Page and F. McGarry, December 1983

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-002, FORTRAN Static Source Code Analyzer Program (SAP) System Description, W. A. Taylor and W. J. Decker, August 1982

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, September 1982

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

¹SEL-82-005, Glossary of Software Engineering Laboratory Terms, M. G. Rohleder, December 1982

SEL-82-105, Glossary of Software Engineering Laboratory Terms, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

¹SEL-82-006, Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, D. N. Card, November 1982

SEL-82-106, Annotated Bibliography of Software Engineering Laboratory Literature, D. N. Card, T. A. Babst, and F. E. McGarry, November 1983

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-83-001, Software Cost Estimation Experiences, F. E. McGarry, G. Page, D. N. Card, et al., November 1983

SEL-83-002, Measures and Metrics for Software Development, D. N. Card, F. E. McGarry, G. Page, et al., November 1983

SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983

SEL-83-004, SEL Data Base Retrieval System (DARES) User's Guide, T. A. Babst and W. J. Decker, November 1983

SEL-83-005, SEL Data Base Retrieval System (DARES) System Description, P. Lo and W. J. Decker, November 1983

SEL-83-006, Monitoring Software Development Through Dynamic Variables, C. W. Doerflinger, November 1983

SEL-83-007, Proceedings From the Eighth Annual Software Engineering Workshop, November 1983

SEL-RELATED LITERATURE

²Agresti, W. W. , F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," Program Transformation and Programming Environments. New York: Springer-Verlag, 1984

³Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

³Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation," University of Maryland, Technical Memorandum, October 1979

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

³Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

³Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

²Basili, V. R., and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, University of Maryland, Technical Report TR-1195, August 1982

³Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

²Basili, V. R., R. W. Selby, and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," IEEE Transactions on Software Engineering, November 1983

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

²Basili, V. R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

³Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

³Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

³Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

²Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

²Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

Card, D. N., and V. E. Church, "Analysis Software Requirements for the Data Retrieval System," Computer Sciences Corporation Technical Memorandum, March 1983

Card, D. N., and V. E. Church, "A Plan of Analysis for Software Engineering Laboratory Data," Computer Sciences Corporation Technical Memorandum, March 1983

Card, D. N., and M. G. Rohleder, "Report of Data Expansion Efforts," Computer Sciences Corporation, Technical Memorandum, September 1982

³Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

²Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," Proceedings of the Seventh International Computer Software and Applications Conference. New York: Computer Societies Press, 1983

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

McGarry, F. E., G. Page, and R. D. Werking, Software Development History of the Dynamics Explorer (DE) Attitude Ground Support System (AGSS), June 1983

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

³Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science.
New York: Computer Societies Press, 1979

²Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings),
November 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

¹This document superseded by revised document.

²This article also appears in SEL-83-003, Collected Software Engineering Papers: Volume II, November 1983.